In [103]:
```python
import os
cwd = os.getcwd()
cwd
```

Out[103]:
```
'C:\\Users\\USER\\Desktop\\bulldozer-price-prediction'
```

Predicting the Sale Price of Bulldozers using Machine Learning

In this notebook, we're going to go through an example machine learning project with the goal of predicting the sale price of bulldozers.

1. Problem definition

> How well can we predict the future sale price of a bulldozer, given its characteristics and previous examples of how much similar bulldozer have been sold for?

2. Data

The data is downloaded from the kaggle Bluebook for Bulldozers competion: https://www.kaggle.com/competitions/bluebook-for-bulldozers/data (https://www.kaggle.com/competitions/bluebook-for-bulldozers/data)

There are 3 main dadsets:

- Train.csv is the training set, which contains data through the end of 2011.
- Valid.csv is the validation set, which contains data from January 1, 2012 - April 30, 2012 You make predictions on this set throughout the majority of the competition. Your score on this set is used to create the public leaderboard.
- Test.csv is the test set, which won't be released until the last week of the competition. It contains data from May 1, 2012 - November 2012. Your score on the test set determines your final rank for the competition.

3. Evaluation

The evaluation metric for this competition is the RMSLE (root mean squared log error) between the actual and predicted auction prices.

For more on the evaluation of this project check: [https://www.kaggle.com/competitions/bluebook-for-bulldozers/overview/evaluation (https://www.kaggle.com/competitions/bluebook-for-bulldozers/overview/evaluation)](https://www.kaggle.com/competitions/bluebook-for-bulldozers/overview/evaluation)

Note: The goal for most regression evaluation metrics is to minimize the error. For example, our goal for this project will be to build a machine learning model which minimizes RMSLE.

4. Features

Kaggle provides a data dictionary detailing all of the features of the dataset. You can view this data dictionary on excel, on kaggle or Google sheets: [https://www.kaggle.com/competitions/bluebook-for-bulldozers/data (https://www.kaggle.com/competitions/bluebook-for-bulldozers/data)](https://www.kaggle.com/competitions/bluebook-for-bulldozers/data)

In [104]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

In [105]:

```
# import training and validation sets
df = pd.read_csv('TrainAndValid.csv', low_memory=False)
```

In [106]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Data columns (total 53 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object
 9   saledate                  412698 non-null  object
 10  fiModelDesc               412698 non-null  object
 11  fiBaseModel               412698 non-null  object
 12  fiSecondaryDesc           271971 non-null  object
 13  fiModelSeries             58667 non-null   object
 14  fiModelDescriptor         74816 non-null   object
 15  ProductSize               196093 non-null  object
 16  fiProductClassDesc        412698 non-null  object
 17  state                     412698 non-null  object
 18  ProductGroup              412698 non-null  object
 19  ProductGroupDesc          412698 non-null  object
 20  Drive_System              107087 non-null  object
 21  Enclosure                 412364 non-null  object
 22  Forks                     197715 non-null  object
 23  Pad_Type                  81096 non-null   object
 24  Ride_Control              152728 non-null  object
 25  Stick                     81096 non-null   object
 26  Transmission              188007 non-null  object
 27  Turbocharged              81096 non-null   object
 28  Blade_Extension           25983 non-null   object
 29  Blade_Width               25983 non-null   object
 30  Enclosure_Type            25983 non-null   object
 31  Engine_Horsepower         25983 non-null   object
 32  Hydraulics                330133 non-null  object
 33  Pushblock                 25983 non-null   object
 34  Ripper                    106945 non-null  object
 35  Scarifier                 25994 non-null   object
 36  Tip_Control               25983 non-null   object
 37  Tire_Size                 97638 non-null   object
```

```
 38  Coupler                   220679 non-null  object
 39  Coupler_System            44974 non-null   object
 40  Grouser_Tracks            44875 non-null   object
 41  Hydraulics_Flow           44875 non-null   object
 42  Track_Type                102193 non-null  object
 43  Undercarriage_Pad_Width   102916 non-null  object
 44  Stick_Length              102261 non-null  object
 45  Thumb                     102332 non-null  object
 46  Pattern_Changer           102261 non-null  object
 47  Grouser_Type              102193 non-null  object
 48  Backhoe_Mounting          80712 non-null   object
 49  Blade_Type                81875 non-null   object
 50  Travel_Controls           81877 non-null   object
 51  Differential_Type         71564 non-null   object
 52  Steering_Controls         71522 non-null   object
dtypes: float64(3), int64(5), object(45)
memory usage: 166.9+ MB
```

In [107]:
```
len(df)
```

Out[107]:
```
412698
```

In [108]:

```
# to check missing values
df.isna().sum()
```

Out[108]:
```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
saledate                         0
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
```

```
Undercarriage_Pad_Width     309782
Stick_Length                310437
Thumb                       310366
Pattern_Changer             310437
Grouser_Type                310505
Backhoe_Mounting            331986
Blade_Type                  330823
Travel_Controls             330821
Differential_Type           341134
Steering_Controls           341176
dtype: int64
```

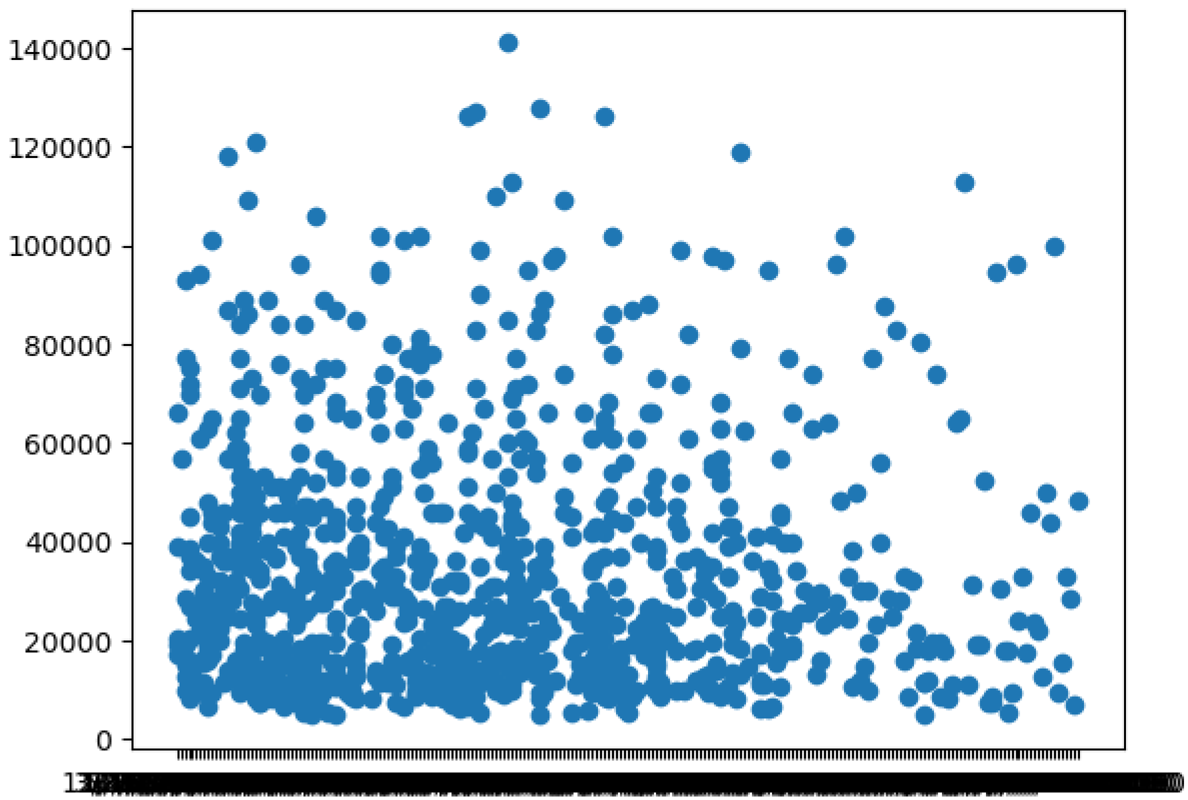
In [109]:

```
# to find the column names
df.columns
```

Out[109]:

```
Index(['SalesID', 'SalePrice', 'MachineID', 'ModelID', 'datasource',
       'auctioneerID', 'YearMade', 'MachineHoursCurrentMeter', 'UsageBand',
       'saledate', 'fiModelDesc', 'fiBaseModel', 'fiSecondaryDesc',
       'fiModelSeries', 'fiModelDescriptor', 'ProductSize',
       'fiProductClassDesc', 'state', 'ProductGroup', 'ProductGroupDesc',
       'Drive_System', 'Enclosure', 'Forks', 'Pad_Type', 'Ride_Control',
       'Stick', 'Transmission', 'Turbocharged', 'Blade_Extension',
       'Blade_Width', 'Enclosure_Type', 'Engine_Horsepower', 'Hydraulics',
       'Pushblock', 'Ripper', 'Scarifier', 'Tip_Control', 'Tire_Size',
       'Coupler', 'Coupler_System', 'Grouser_Tracks', 'Hydraulics_Flow',
       'Track_Type', 'Undercarriage_Pad_Width', 'Stick_Length', 'Thumb',
       'Pattern_Changer', 'Grouser_Type', 'Backhoe_Mounting', 'Blade_Type',
       'Travel_Controls', 'Differential_Type', 'Steering_Controls'],
      dtype='object')
```

In [110]:
```
fig, ax = plt.subplots()
ax.scatter(df['saledate'][:1000], df['SalePrice'][: 1000]);
```



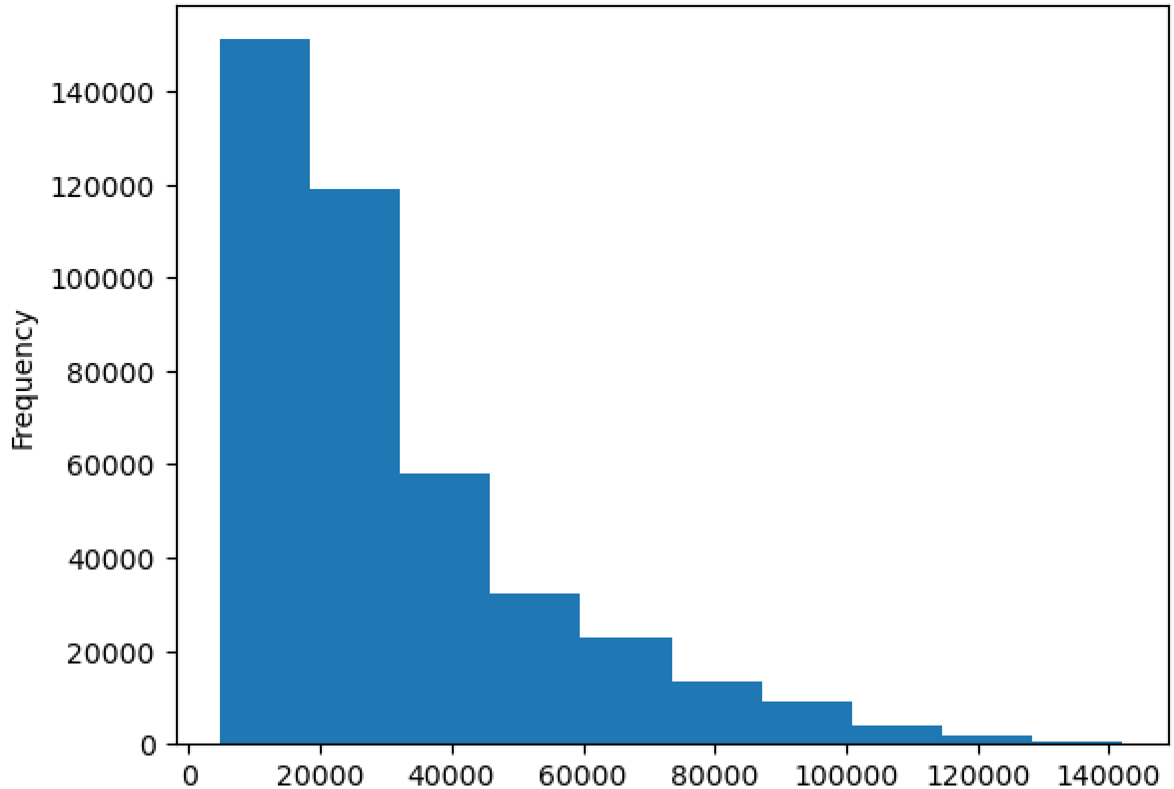
In [111]:
```
df.saledate[: 1000]
```

Out[111]:
```
0      11/16/2006 0:00
1       3/26/2004 0:00
2       2/26/2004 0:00
3       5/19/2011 0:00
4       7/23/2009 0:00
             ...
995     7/16/2009 0:00
996     6/14/2007 0:00
997     9/22/2005 0:00
998     7/28/2005 0:00
999     6/16/2011 0:00
Name: saledate, Length: 1000, dtype: object
```

In [112]:
```
df.saledate.dtypes
```

Out[112]:
```
dtype('O')
```

In [113]:
```
df.SalePrice.plot.hist();
```



Parsing dates

When we work with time series data, we want to enrich the time & date component as much as possible.

we can do that by telling pandas which of our columns has date in it using the `parse_date` parameter.

In [114]:
```
# import data again but this time parse dates

df = pd.read_csv('TrainAndValid.csv',
                 low_memory=False,
                parse_dates=['saledate'])
```

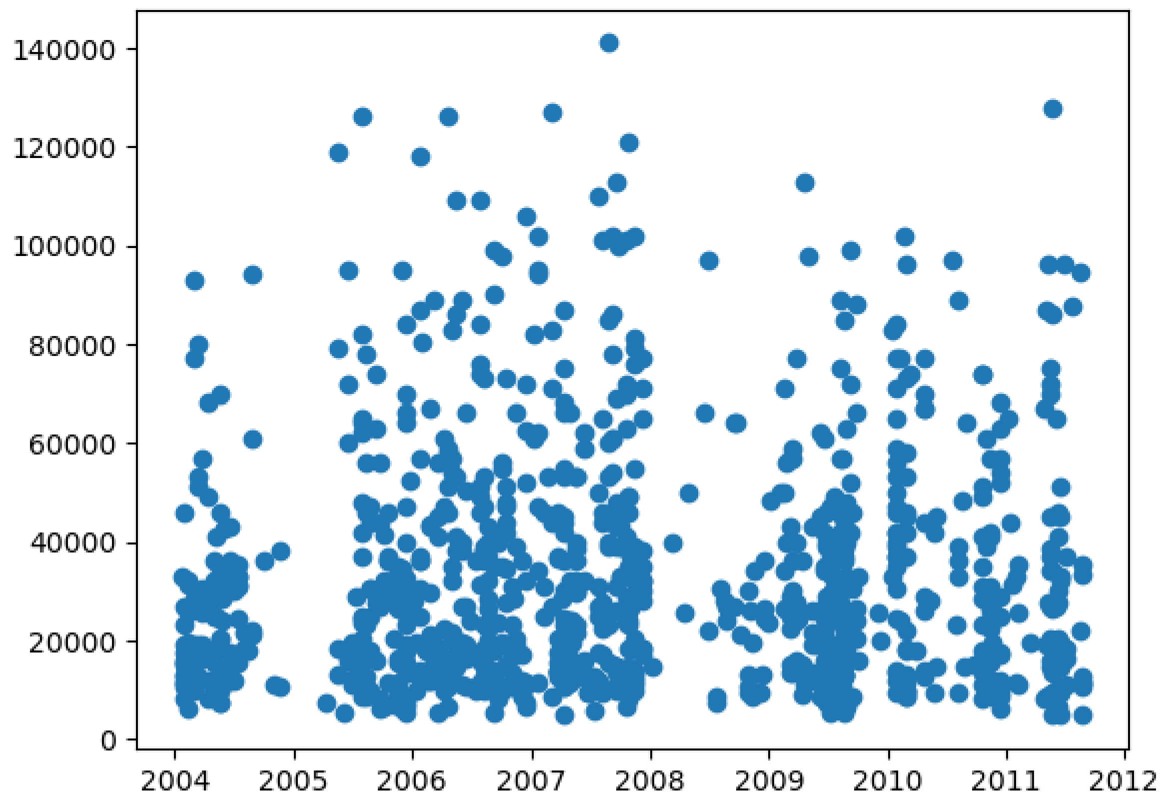
In [115]:
```
df.saledate.dtypes
```

Out[115]:
```
dtype('<M8[ns]')
```

In [116]:
```
df.saledate[: 1000]
```

Out[116]:
```
0     2006-11-16
1     2004-03-26
2     2004-02-26
3     2011-05-19
4     2009-07-23
         ...
995   2009-07-16
996   2007-06-14
997   2005-09-22
998   2005-07-28
999   2011-06-16
Name: saledate, Length: 1000, dtype: datetime64[ns]
```

In [117]:
```
fig, ax = plt.subplots()
ax.scatter(df['saledate'][: 1000], df['SalePrice'][: 1000]);
```



In [118]:

```
df.head()
```

Out[118]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate	...
0	1139246	66000.0	999089	3157	121	3.0	2004	68.0	Low	2006-11-16	...
1	1139248	57000.0	117657	77	121	3.0	1996	4640.0	Low	2004-03-26	...
2	1139249	10000.0	434808	7009	121	3.0	2001	2838.0	High	2004-02-26	...
3	1139251	38500.0	1026470	332	121	3.0	2001	3486.0	High	2011-05-19	...
4	1139253	11000.0	1057373	17311	121	3.0	2007	722.0	Medium	2009-07-23	...

5 rows × 53 columns

In [119]:
```
df.head().T
```

Out[119]:

	0	1	2	3	4
SalesID	1139246	1139248	1139249	1139251	1139253
SalePrice	66000.0	57000.0	10000.0	38500.0	11000.0
MachineID	999089	117657	434808	1026470	1057373
ModelID	3157	77	7009	332	17311
datasource	121	121	121	121	121
auctioneerID	3.0	3.0	3.0	3.0	3.0
YearMade	2004	1996	2001	2001	2007
MachineHoursCurrentMeter	68.0	4640.0	2838.0	3486.0	722.0
UsageBand	Low	Low	High	High	Medium
saledate	2006-11-16 00:00:00	2004-03-26 00:00:00	2004-02-26 00:00:00	2011-05-19 00:00:00	2009-07-23 00:00:00
fiModelDesc	521D	950FII	226	PC120-6E	S175
fiBaseModel	521	950	226	PC120	S175
fiSecondaryDesc	D	F	NaN	NaN	NaN
fiModelSeries	NaN	II	NaN	-6E	NaN
fiModelDescriptor	NaN	NaN	NaN	NaN	NaN
ProductSize	NaN	Medium	NaN	Small	NaN
fiProductClassDesc	Wheel Loader - 110.0 to 120.0 Horsepower	Wheel Loader - 150.0 to 175.0 Horsepower	Skid Steer Loader - 1351.0 to 1601.0 Lb Operat...	Hydraulic Excavator, Track - 12.0 to 14.0 Metr...	Skid Steer Loader - 1601.0 to 1751.0 Lb Operat...
state	Alabama	North Carolina	New York	Texas	New York
ProductGroup	WL	WL	SSL	TEX	SSL
ProductGroupDesc	Wheel Loader	Wheel Loader	Skid Steer Loaders	Track Excavators	Skid Steer Loaders
Drive_System	NaN	NaN	NaN	NaN	NaN
Enclosure	EROPS w AC	EROPS w AC	OROPS	EROPS w AC	EROPS
Forks	None or Unspecified	None or Unspecified	None or Unspecified	NaN	None or Unspecified
Pad_Type	NaN	NaN	NaN	NaN	NaN

	0	1	2	3	4
Ride_Control	None or Unspecified	None or Unspecified	NaN	NaN	NaN
Stick	NaN	NaN	NaN	NaN	NaN
Transmission	NaN	NaN	NaN	NaN	NaN
Turbocharged	NaN	NaN	NaN	NaN	NaN
Blade_Extension	NaN	NaN	NaN	NaN	NaN
Blade_Width	NaN	NaN	NaN	NaN	NaN
Enclosure_Type	NaN	NaN	NaN	NaN	NaN
Engine_Horsepower	NaN	NaN	NaN	NaN	NaN
Hydraulics	2 Valve	2 Valve	Auxiliary	2 Valve	Auxiliary
Pushblock	NaN	NaN	NaN	NaN	NaN
Ripper	NaN	NaN	NaN	NaN	NaN
Scarifier	NaN	NaN	NaN	NaN	NaN
Tip_Control	NaN	NaN	NaN	NaN	NaN
Tire_Size	None or Unspecified	23.5	NaN	NaN	NaN
Coupler	None or Unspecified	None or Unspecified	None or Unspecified	None or Unspecified	None or Unspecified
Coupler_System	NaN	NaN	None or Unspecified	NaN	None or Unspecified
Grouser_Tracks	NaN	NaN	None or Unspecified	NaN	None or Unspecified
Hydraulics_Flow	NaN	NaN	Standard	NaN	Standard
Track_Type	NaN	NaN	NaN	NaN	NaN
Undercarriage_Pad_Width	NaN	NaN	NaN	NaN	NaN
Stick_Length	NaN	NaN	NaN	NaN	NaN
Thumb	NaN	NaN	NaN	NaN	NaN
Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN
Backhoe_Mounting	NaN	NaN	NaN	NaN	NaN
Blade_Type	NaN	NaN	NaN	NaN	NaN

	0	1	2	3	4
Travel_Controls	NaN	NaN	NaN	NaN	NaN
Differential_Type	Standard	Standard	NaN	NaN	NaN
Steering_Controls	Conventional	Conventional	NaN	NaN	NaN

In [120]:

```
df.saledate.head(20)
```

Out[120]:

```
0    2006-11-16
1    2004-03-26
2    2004-02-26
3    2011-05-19
4    2009-07-23
5    2008-12-18
6    2004-08-26
7    2005-11-17
8    2009-08-27
9    2007-08-09
10   2008-08-21
11   2006-08-24
12   2005-10-20
13   2006-01-26
14   2006-01-03
15   2006-11-16
16   2007-06-14
17   2010-01-28
18   2006-03-09
19   2005-11-17
Name: saledate, dtype: datetime64[ns]
```

Sort DataFrame by saledate

when working with time series data, it's a good idea to sort it by date.

In [121]:
```
# sort DataFrame in date order
df.sort_values(by=['saledate'], inplace=True, ascending=True)
df.saledate.head(20)
```

Out[121]:
```
205615   1989-01-17
274835   1989-01-31
141296   1989-01-31
212552   1989-01-31
62755    1989-01-31
54653    1989-01-31
81383    1989-01-31
204924   1989-01-31
135376   1989-01-31
113390   1989-01-31
113394   1989-01-31
116419   1989-01-31
32138    1989-01-31
127610   1989-01-31
76171    1989-01-31
127000   1989-01-31
128130   1989-01-31
127626   1989-01-31
55455    1989-01-31
55454    1989-01-31
Name: saledate, dtype: datetime64[ns]
```

make a copy of the original DataFrame

We make a copy of the original dataframe so when we manipulate the copy, we've still got our original data.

In [122]:
```
# make a copy
df_tmp = df.copy()
```

In [123]:
```
df_tmp.saledate.head(20)
```

Out[123]:
```
205615   1989-01-17
274835   1989-01-31
141296   1989-01-31
212552   1989-01-31
62755    1989-01-31
54653    1989-01-31
81383    1989-01-31
204924   1989-01-31
135376   1989-01-31
113390   1989-01-31
113394   1989-01-31
116419   1989-01-31
32138    1989-01-31
127610   1989-01-31
76171    1989-01-31
127000   1989-01-31
128130   1989-01-31
127626   1989-01-31
55455    1989-01-31
55454    1989-01-31
Name: saledate, dtype: datetime64[ns]
```

Feature Engineering

Add datetime parameters for `saledate` column

In [124]:
```
df_tmp[: 1].saledate.dt.year
```

Out[124]:
```
205615    1989
Name: saledate, dtype: int64
```

In [125]:
```
df_tmp[: 1].saledate.dt.day
```

Out[125]:
```
205615    17
Name: saledate, dtype: int64
```

In [126]:
```
df_tmp[: 1].saledate.dt.month
```

Out[126]:
```
205615    1
Name: saledate, dtype: int64
```

In [127]:
```
df_tmp[:1].saledate
```

Out[127]:
```
205615   1989-01-17
Name: saledate, dtype: datetime64[ns]
```

In [128]:
```
df_tmp['saleYear'] = df_tmp.saledate.dt.year
df_tmp['saleMonth'] = df_tmp.saledate.dt.month
df_tmp['saleDay'] = df_tmp.saledate.dt.day
df_tmp['saleDayOfWeek'] = df_tmp.saledate.dt.dayofweek
df_tmp['saleDayOfYear'] = df_tmp.saledate.dt.dayofyear
```

In [129]:
```
df_tmp.head().T
```

Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN
Backhoe_Mounting	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Blade_Type	Straight	NaN	Straight	NaN	PAT
Travel_Controls	None or Unspecified	NaN	None or Unspecified	NaN	Lever
Differential_Type	NaN	Standard	NaN	Standard	NaN
Steering_Controls	NaN	Conventional	NaN	Conventional	NaN
saleYear	1989	1989	1989	1989	1989
saleMonth	1	1	1	1	1
saleDay	17	31	31	31	31
saleDayOfWeek	1	1	1	1	1
saleDayOfYear	17	31	31	31	31

In [130]:
```
# Now we've enrich our DataFrame with date time features, we can remove 'saledate'
df_tmp.drop('saledate', axis = 1, inplace=True)
```

In [131]:
```
# check the values of different columns
df_tmp.state.value_counts()
```

Out[131]:

```
Florida           67320
Texas             53110
California        29761
Washington        16222
Georgia           14633
Maryland          13322
Mississippi       13240
Ohio              12369
Illinois          11540
Colorado          11529
New Jersey        11156
North Carolina    10636
Tennessee         10298
Alabama           10292
Pennsylvania      10234
South Carolina     9951
Arizona            9364
New York           8639
Connecticut        8276
Minnesota          7885
Missouri           7178
Nevada             6932
Louisiana          6627
Kentucky           5351
Maine              5096
Indiana            4124
Arkansas           3933
New Mexico         3631
Utah               3046
Unspecified        2801
Wisconsin          2745
New Hampshire      2738
Virginia           2353
Idaho              2025
Oregon             1911
Michigan           1831
Wyoming            1672
Montana            1336
Iowa               1336
Oklahoma           1326
Nebraska            866
West Virginia       840
Kansas              667
```

```
Delaware           510
North Dakota       480
Alaska             430
Massachusetts      347
Vermont            300
South Dakota       244
Hawaii             118
Rhode Island        83
Puerto Rico         42
Washington DC        2
Name: state, dtype: int64
```

5. Modelling

we've done enough EDA (we could always do more) but let's start to do some model-driven EDA.

In [132]:

```
df_tmp.head()
```

Out[132]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelID
205615	1646770	9500.0	1126363	8434	132	18.0	1974	NaN	NaN	TI
274835	1821514	14000.0	1194089	10150	132	99.0	1980	NaN	NaN	
141296	1505138	50000.0	1473654	4139	132	99.0	1978	NaN	NaN	[
212552	1671174	16000.0	1327630	8591	132	99.0	1980	NaN	NaN	
62755	1329056	22000.0	1336053	4089	132	99.0	1984	NaN	NaN	[

5 rows × 57 columns

In [133]:

```
# let's build a machine learning model
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_jobs=-1,
                              random_state=42)
model.fit(df_tmp.drop('SalePrice', axis = 1), df_tmp['SalePrice'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6476\4172265634.py in <module>
      4 model = RandomForestRegressor(n_jobs=-1,
      5                               random_state=42)
----> 6 model.fit(df_tmp.drop('SalePrice', axis = 1), df_tmp['SalePrice'])

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sample_weight)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is not supported.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    329         )

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963 
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062 
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
```

```
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'Low'
```

In [134]:
```
# we have to convert strings to numbers before our ML can work
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   SalesID                   412698 non-null  int64  
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64  
 3   ModelID                   412698 non-null  int64  
 4   datasource                412698 non-null  int64  
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64  
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object 
 9   fiModelDesc               412698 non-null  object 
 10  fiBaseModel               412698 non-null  object 
 11  fiSecondaryDesc           271971 non-null  object 
 12  fiModelSeries             58667 non-null   object 
 13  fiModelDescriptor         74816 non-null   object 
 14  ProductSize               196093 non-null  object 
 15  fiProductClassDesc        412698 non-null  object 
 16  state                     412698 non-null  object 
 17  ProductGroup              412698 non-null  object 
 18  ProductGroupDesc          412698 non-null  object 
 19  Drive_System              107087 non-null  object 
 20  Enclosure                 412364 non-null  object 
 21  Forks                     197715 non-null  object 
 22  Pad_Type                  81096 non-null   object 
 23  Ride_Control              152728 non-null  object 
 24  Stick                     81096 non-null   object 
 25  Transmission              188007 non-null  object 
 26  Turbocharged              81096 non-null   object 
 27  Blade_Extension           25983 non-null   object 
 28  Blade_Width               25983 non-null   object 
 29  Enclosure_Type            25983 non-null   object 
 30  Engine_Horsepower         25983 non-null   object 
 31  Hydraulics                330133 non-null  object 
 32  Pushblock                 25983 non-null   object 
 33  Ripper                    106945 non-null  object 
 34  Scarifier                 25994 non-null   object 
 35  Tip_Control               25983 non-null   object 
 36  Tire_Size                 97638 non-null   object 
 37  Coupler                   220679 non-null  object 
```

```
 38  Coupler_System            44974 non-null   object
 39  Grouser_Tracks            44875 non-null   object
 40  Hydraulics_Flow           44875 non-null   object
 41  Track_Type                102193 non-null  object
 42  Undercarriage_Pad_Width   102916 non-null  object
 43  Stick_Length              102261 non-null  object
 44  Thumb                     102332 non-null  object
 45  Pattern_Changer           102261 non-null  object
 46  Grouser_Type              102193 non-null  object
 47  Backhoe_Mounting          80712 non-null   object
 48  Blade_Type                81875 non-null   object
 49  Travel_Controls           81877 non-null   object
 50  Differential_Type         71564 non-null   object
 51  Steering_Controls         71522 non-null   object
 52  saleYear                  412698 non-null  int64
 53  saleMonth                 412698 non-null  int64
 54  saleDay                   412698 non-null  int64
 55  saleDayOfWeek             412698 non-null  int64
 56  saleDayOfYear             412698 non-null  int64
dtypes: float64(3), int64(10), object(44)
memory usage: 182.6+ MB
```

In [135]:
```
# A lot of missing data
df_tmp.isna().sum()
```

Out[135]:
```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
Undercarriage_Pad_Width     309782
```

```
Stick_Length                310437
Thumb                       310366
Pattern_Changer             310437
Grouser_Type                310505
Backhoe_Mounting            331986
Blade_Type                  330823
Travel_Controls             330821
Differential_Type           341134
Steering_Controls           341176
saleYear                         0
saleMonth                        0
saleDay                          0
saleDayOfWeek                    0
saleDayOfYear                    0
dtype: int64
```

Convert string to categories

One way we can turn all of our data into numbers is by converting them into pandas categories.

In [136]:
```
df_tmp.head().T
```

Out[136]:

	205615	274835	141296	212552	62755
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
auctioneerID	18.0	99.0	99.0	99.0	99.0
YearMade	1974	1980	1978	1980	1984
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	NaN
UsageBand	NaN	NaN	NaN	NaN	NaN
fiModelDesc	TD20	A66	D7G	A62	D3B
fiBaseModel	TD20	A66	D7	A62	D3

In [137]:
```
pd.api.types.is_string_dtype(df_tmp['UsageBand'])
```

Out[137]:
```
True
```

In [138]:
```
# Find the columns which contain strings
for label, content in df_tmp.items():
    if pd.api.types.is_string_dtype(content):
        print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
```

```
Differential_Type
Steering_Controls
```

In [139]:

```python
# if you're wondering what df.items() does, here's an example
random_dict = {'key1': 'hello',
               'key2': 'world!'}
for key, value in random_dict.items():
    print(f'this is a key: {key}',
          f'this is a value: {value}' )
```

```
this is a key: key1 this is a value: hello
this is a key: key2 this is a value: world!
```

In [140]:

```python
# This will turn all of the string value into category values
for label, content in df_tmp.items():
    if pd.api.types.is_string_dtype(content):
        df_tmp[label] = content.astype('category').cat.as_ordered()
```

In [141]:
```
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
 #   Column                    Non-Null Count   Dtype   
---  ------                    --------------   -----   
 0   SalesID                   412698 non-null  int64   
 1   SalePrice                 412698 non-null  float64 
 2   MachineID                 412698 non-null  int64   
 3   ModelID                   412698 non-null  int64   
 4   datasource                412698 non-null  int64   
 5   auctioneerID              392562 non-null  float64 
 6   YearMade                  412698 non-null  int64   
 7   MachineHoursCurrentMeter  147504 non-null  float64 
 8   UsageBand                 73670 non-null   category
 9   fiModelDesc               412698 non-null  category
 10  fiBaseModel               412698 non-null  category
 11  fiSecondaryDesc           271971 non-null  category
 12  fiModelSeries             58667 non-null   category
 13  fiModelDescriptor         74816 non-null   category
 14  ProductSize               196093 non-null  category
 15  fiProductClassDesc        412698 non-null  category
 16  state                     412698 non-null  category
 17  ProductGroup              412698 non-null  category
 18  ProductGroupDesc          412698 non-null  category
 19  Drive_System              107087 non-null  category
 20  Enclosure                 412364 non-null  category
 21  Forks                     197715 non-null  category
 22  Pad_Type                  81096 non-null   category
 23  Ride_Control              152728 non-null  category
 24  Stick                     81096 non-null   category
 25  Transmission              188007 non-null  category
 26  Turbocharged              81096 non-null   category
 27  Blade_Extension           25983 non-null   category
 28  Blade_Width               25983 non-null   category
 29  Enclosure_Type            25983 non-null   category
 30  Engine_Horsepower         25983 non-null   category
 31  Hydraulics                330133 non-null  category
 32  Pushblock                 25983 non-null   category
 33  Ripper                    106945 non-null  category
 34  Scarifier                 25994 non-null   category
 35  Tip_Control               25983 non-null   category
 36  Tire_Size                 97638 non-null   category
 37  Coupler                   220679 non-null  category
```

```
 38  Coupler_System           44974 non-null   category
 39  Grouser_Tracks           44875 non-null   category
 40  Hydraulics_Flow          44875 non-null   category
 41  Track_Type               102193 non-null  category
 42  Undercarriage_Pad_Width  102916 non-null  category
 43  Stick_Length             102261 non-null  category
 44  Thumb                    102332 non-null  category
 45  Pattern_Changer          102261 non-null  category
 46  Grouser_Type             102193 non-null  category
 47  Backhoe_Mounting         80712 non-null   category
 48  Blade_Type               81875 non-null   category
 49  Travel_Controls          81877 non-null   category
 50  Differential_Type        71564 non-null   category
 51  Steering_Controls        71522 non-null   category
 52  saleYear                 412698 non-null  int64
 53  saleMonth                412698 non-null  int64
 54  saleDay                  412698 non-null  int64
 55  saleDayOfWeek            412698 non-null  int64
 56  saleDayOfYear            412698 non-null  int64
dtypes: category(44), float64(3), int64(10)
memory usage: 63.2 MB
```

In [142]:

```
df_tmp.state.cat.categories
```

Out[142]:

```
Index(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
       'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Puerto Rico', 'Rhode Island', 'South Carolina',
       'South Dakota', 'Tennessee', 'Texas', 'Unspecified', 'Utah', 'Vermont',
       'Virginia', 'Washington', 'Washington DC', 'West Virginia', 'Wisconsin',
       'Wyoming'],
      dtype='object')
```

In [143]:
```
df_tmp.state.cat.codes
```

Out[143]:
```
205615    43
274835     8
141296     8
212552     8
62755      8
          ..
410879     4
412476     4
411927     4
407124     4
409203     4
Length: 412698, dtype: int8
```

Thanks to pandas categories, we now have a way to access all of our data in the form of numbers.

But we still have a bunch of missing data...

In [144]:
```
df_tmp.isnull().sum()/len(df_tmp)
```

Out[144]:
```
SalesID                     0.000000
SalePrice                   0.000000
MachineID                   0.000000
ModelID                     0.000000
datasource                  0.000000
auctioneerID                0.048791
YearMade                    0.000000
MachineHoursCurrentMeter    0.642586
UsageBand                   0.821492
fiModelDesc                 0.000000
fiBaseModel                 0.000000
fiSecondaryDesc             0.340993
fiModelSeries               0.857845
fiModelDescriptor           0.818715
ProductSize                 0.524851
fiProductClassDesc          0.000000
state                       0.000000
ProductGroup                0.000000
ProductGroupDesc            0.000000
Drive_System                0.740520
Enclosure                   0.000809
Forks                       0.520921
Pad_Type                    0.803498
Ride_Control                0.629928
Stick                       0.803498
Transmission                0.544444
Turbocharged                0.803498
Blade_Extension             0.937041
Blade_Width                 0.937041
Enclosure_Type              0.937041
Engine_Horsepower           0.937041
Hydraulics                  0.200062
Pushblock                   0.937041
Ripper                      0.740864
Scarifier                   0.937014
Tip_Control                 0.937041
Tire_Size                   0.763415
Coupler                     0.465277
Coupler_System              0.891024
Grouser_Tracks              0.891264
Hydraulics_Flow             0.891264
Track_Type                  0.752378
Undercarriage_Pad_Width     0.750626
```

```
Stick_Length             0.752213
Thumb                    0.752041
Pattern_Changer          0.752213
Grouser_Type             0.752378
Backhoe_Mounting         0.804428
Blade_Type               0.801610
Travel_Controls          0.801606
Differential_Type        0.826595
Steering_Controls        0.826697
saleYear                 0.000000
saleMonth                0.000000
saleDay                  0.000000
saleDayOfWeek            0.000000
saleDayOfYear            0.000000
dtype: float64
```

save preprocessed data

In [145]:

```
# eExport current tmp dataframe
df_tmp.to_csv('train_tmp.csv',
             index=False)
```

In [146]:

```
# Import preprocessed data

df_tmp = pd.read_csv('train_tmp.csv',
                     low_memory=False)
df_tmp.head().T
```

Out[146]:

	0	1	2	3	4
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
auctioneerID	18.0	99.0	99.0	99.0	99.0
YearMade	1974	1980	1978	1980	1984
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	NaN
UsageBand	NaN	NaN	NaN	NaN	NaN
fiModelDesc	TD20	A66	D7G	A62	D3B
fiBaseModel	TD20	A66	D7	A62	D3

In [147]:
```
df_tmp.isna().sum()
```

Out[147]:
```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
Undercarriage_Pad_Width     309782
```

```
Stick_Length                310437
Thumb                       310366
Pattern_Changer             310437
Grouser_Type                310505
Backhoe_Mounting            331986
Blade_Type                  330823
Travel_Controls             330821
Differential_Type           341134
Steering_Controls           341176
saleYear                         0
saleMonth                        0
saleDay                          0
saleDayOfWeek                    0
saleDayOfYear                    0
dtype: int64
```

Fill missing values

Fill numerical missing values first

In [148]:

```
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
SalesID
SalePrice
MachineID
ModelID
datasource
auctioneerID
YearMade
MachineHoursCurrentMeter
saleYear
saleMonth
saleDay
saleDayOfWeek
saleDayOfYear
```

In [149]:
```
df_tmp.ModelID
```

Out[149]:
```
0          8434
1         10150
2          4139
3          8591
4          4089
          ...
412693     5266
412694    19330
412695    17244
412696     3357
412697     4701
Name: ModelID, Length: 412698, dtype: int64
```

In [150]:
```
# check for which numeric clumns have null values
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)

```

```
auctioneerID
MachineHoursCurrentMeter
```

In [151]:
```
# Fill numeric rows with the median
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Add a binary column which tells us if the data was missing or not
            df_tmp[label+'_is_missing'] = pd.isnull(content)
            # Fill missing numeric values with median
            df_tmp[label] = content.fillna(content.median())

```

In [152]:
```
# Demostrate how median is more robust than mean
hundreds = np.full((1000), 100)
hundreds_billion = np.append(hundreds, 1000000000)
np.mean(hundreds), np.mean(hundreds_billion), np.median(hundreds), np.median(hundreds_billion)
```

Out[152]:
```
(100.0, 999100.8991008991, 100.0, 100.0)
```

In [153]:
```
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

In [154]:
```
# cheeck to see how many examples were missing
df_tmp.auctioneerID_is_missing.value_counts(), df_tmp.MachineHoursCurrentMeter_is_missing.value_count
```

Out[154]:
```
(False    392562
 True      20136
 Name: auctioneerID_is_missing, dtype: int64,
 True     265194
 False    147504
 Name: MachineHoursCurrentMeter_is_missing, dtype: int64)
```

In [155]:
```
df_tmp.isna().sum()
```

Out[155]:
```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                     0
YearMade                         0
MachineHoursCurrentMeter         0
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
Undercarriage_Pad_Width     309782
```

```
Stick_Length                          310437
Thumb                                 310366
Pattern_Changer                       310437
Grouser_Type                          310505
Backhoe_Mounting                      331986
Blade_Type                            330823
Travel_Controls                       330821
Differential_Type                     341134
Steering_Controls                     341176
saleYear                                   0
saleMonth                                  0
saleDay                                    0
saleDayOfWeek                              0
saleDayOfYear                              0
auctioneerID_is_missing                    0
MachineHoursCurrentMeter_is_missing        0
dtype: int64
```

Filling and turning categorical variables into numbers

In [156]:
```
# check for columns which arem't numeric
for label, content in df_tmp.items():
   if not pd.api.types.is_numeric_dtype(content):
    print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
```

```
Differential_Type
Steering_Controls
```

In [157]:

```python
# Turn categorical variables into numbers and fill missing
for label, content in df_tmp.items():
    if not pd.api.types.is_numeric_dtype(content):
        # Add binary column to indicate whether sample has missing value
        df_tmp[label+'_is_missing'] = pd.isnull(content)
        # Turn categories into numbers and add +1
        df_tmp[label] = pd.Categorical(content).codes + 1
```

In [158]:

```python
pd.Categorical(df_tmp['state']).codes
```

Out[158]:

```
array([43,  8,  8, ...,  4,  4,  4], dtype=int8)
```

In [159]:

```python
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Columns: 103 entries, SalesID to Steering_Controls_is_missing
dtypes: bool(46), float64(3), int16(4), int64(10), int8(40)
memory usage: 77.9 MB
```

In [160]:
```
df_tmp.head().T
```

Out[160]:

	0	1	2	3	4
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
...	...	...	...	...	...
Backhoe_Mounting_is_missing	False	True	False	True	False
Blade_Type_is_missing	False	True	False	True	False
Travel_Controls_is_missing	False	True	False	True	False
Differential_Type_is_missing	True	False	True	False	True
Steering_Controls_is_missing	True	False	True	False	True

103 rows × 5 columns

In [161]:
```
df_tmp.isna().sum()
```

Out[161]:
```
SalesID                         0
SalePrice                       0
MachineID                       0
ModelID                         0
datasource                      0
                               ..
Backhoe_Mounting_is_missing     0
Blade_Type_is_missing           0
Travel_Controls_is_missing      0
Differential_Type_is_missing    0
Steering_Controls_is_missing    0
Length: 103, dtype: int64
```

In [162]:
```
pd.Categorical(df_tmp['UsageBand']).codes
```
Out[162]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int8)
```

In [163]:
```
pd.Categorical(df_tmp['UsageBand']).codes + 1
```
Out[163]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int8)
```

Now that all of data is numeric as well as our dataframe has no missing values, we should be able to build a machine learning model

In [164]:
```
df_tmp.head()
```
Out[164]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	...
0	1646770	9500.0	1126363	8434	132	18.0	1974	0.0	0	4593	...
1	1821514	14000.0	1194089	10150	132	99.0	1980	0.0	0	1820	...
2	1505138	50000.0	1473654	4139	132	99.0	1978	0.0	0	2348	...
3	1671174	16000.0	1327630	8591	132	99.0	1980	0.0	0	1819	...
4	1329056	22000.0	1336053	4089	132	99.0	1984	0.0	0	2119	...

5 rows × 103 columns

In [165]:
```
len(df_tmp)
```
Out[165]:
```
412698
```

In [166]:

```
%%time
# Instantiate model
model = RandomForestRegressor(n_jobs=-1,
                              random_state=42)

# Fit the model
model.fit(df_tmp.drop('SalePrice', axis = 1), df_tmp['SalePrice'])
```

```
Wall time: 10min 25s
```

Out[166]:

```
RandomForestRegressor(n_jobs=-1, random_state=42)
```

In [167]:

```
model.score(df_tmp.drop('SalePrice', axis = 1), df_tmp['SalePrice'])
```

Out[167]:

```
0.9875468079970562
```

Question: Why doesn't the above metric hold water? (why isn't metric reliable)

Splitting data into training/validation sets

In [168]:

```
df_tmp.saleYear
```

Out[168]:

```
0         1989
1         1989
2         1989
3         1989
4         1989
          ...
412693    2012
412694    2012
412695    2012
412696    2012
412697    2012
Name: saleYear, Length: 412698, dtype: int64
```

In [169]:

```
df_tmp.saleYear.value_counts()
```

Out[169]:

```
2009    43849
2008    39767
2011    35197
2010    33390
2007    32208
2006    21685
2005    20463
2004    19879
2001    17594
2000    17415
2002    17246
2003    15254
1998    13046
1999    12793
2012    11573
1997     9785
1996     8829
1995     8530
1994     7929
1993     6303
1992     5519
1991     5109
1989     4806
1990     4529
Name: saleYear, dtype: int64
```

In [170]:

```
# split data into training and validation
df_val = df_tmp[df_tmp.saleYear == 2012]
df_train =df_tmp[df_tmp.saleYear != 2012]

len(df_val), len(df_train)
```

Out[170]:

```
(11573, 401125)
```

In [171]:

```
# split data into x and y
x_train, y_train = df_train.drop('SalePrice', axis = 1), df_train.SalePrice
x_valid, y_valid = df_val.drop('SalePrice', axis = 1), df_val.SalePrice

x_train.shape, y_train.shape, x_valid.shape, y_valid.shape
```

Out[171]:

```
((401125, 102), (401125,), (11573, 102), (11573,))
```

In [172]:

```
y_train
```

Out[172]:

```
0          9500.0
1         14000.0
2         50000.0
3         16000.0
4         22000.0
           ...
401120    29000.0
401121    11000.0
401122    11000.0
401123    18000.0
401124    13500.0
Name: SalePrice, Length: 401125, dtype: float64
```

In [173]:

```
x_train
```

Out[173]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBase
0	1646770	1126363	8434	132	18.0	1974	0.0	0	4593	
1	1821514	1194089	10150	132	99.0	1980	0.0	0	1820	
2	1505138	1473654	4139	132	99.0	1978	0.0	0	2348	
3	1671174	1327630	8591	132	99.0	1980	0.0	0	1819	
4	1329056	1336053	4089	132	99.0	1984	0.0	0	2119	
...	...	...	...	...	...	...	...	...	...	
401120	6260687	1074871	4331	149	2.0	1000	0.0	0	3137	
401121	6312170	1812622	9580	149	2.0	2005	0.0	0	4514	
401122	6312727	1811599	9580	149	2.0	2005	0.0	0	4514	
401123	6315051	1858173	17432	149	2.0	2004	0.0	0	3389	
401124	6260878	1799594	4102	149	2.0	1000	0.0	0	2161	

401125 rows × 102 columns

In [174]:
```
x_valid
```

Out[174]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBase
401125	4449186	2318824	26964	173	99.0	1996	0.0	0	2269	
401126	1222855	531393	23926	121	3.0	1000	8145.0	2	85	
401127	6258613	1810917	13260	149	99.0	2000	24.0	2	1115	
401128	6282680	1543404	1830	149	99.0	2004	4373.0	3	64	
401129	6282759	1863077	11390	149	99.0	2006	3467.0	3	139	
...	...	...	...	...	...	...	...	...	...	
412693	6302984	1915521	5266	149	99.0	2001	0.0	0	2101	
412694	6324811	1919104	19330	149	99.0	2004	0.0	0	240	
412695	6313029	1918416	17244	149	99.0	2004	0.0	0	627	
412696	6266251	509560	3357	149	99.0	1993	0.0	0	83	
412697	6283635	1869284	4701	149	99.0	1000	0.0	0	989	

11573 rows × 102 columns

In [175]:
```
y_valid
```

Out[175]:
```
401125    46173.2
401126    66000.0
401127    26800.0
401128    42100.0
401129    62100.0
            ...
412693    16000.0
412694     6000.0
412695    16000.0
412696    55000.0
412697    34000.0
Name: SalePrice, Length: 11573, dtype: float64
```

Building an evaluation function

In [ ]:

```
# create evaluation function (the competition uses RMSLE)
from sklearn.metrics import mean_squared_log_error, mean_absolute_error, r2_score

def rmsle(y_test, y_preds):
    '''
    calculate root mean squared log error between predictions and true labels'''
    return np.sqrt(mean_squared_log_error(y_test, y_preds))


# create function to evaluate model on a few different levels
def show_scores(model):
    train_preds = model.predict(x_train)
    val_preds = model.predict(x_valid)
    scores = {'Training MAE': mean_absolute_error(y_train, train_preds),
              'Valid MAE': mean_absolute_error(y_valid, val_preds),
              'Training RMSLE':  rmsle(y_train, train_preds),
              'Valid RMSLE': rmsle(y_valid, val_preds),
              'Training R^2': r2_score(y_train, train_preds),
              'Valid R^2': r2_score(y_valid, val_preds)}
    return scores
```

Testing our model on a subset (to tune the hyperparameters)

In [ ]:

```
# # This takes far too long... for experimenting
# %%time
# model = RandomForestRegressor(n_jobs= -1,
#                               random_state=42)
# # model.fit(x_train, y_train)
```

In [ ]:

```
len(x_train)
```

In [ ]:
```
# Change max_samples value
model = RandomForestRegressor(n_jobs=-1,
                              random_state=42,
                              max_samples=10000)
```

In [ ]:
```
%%time
# cutting down on the max number of samples each estimator can see improves training time
model.fit(x_train, y_train)
```

In [ ]:
```
show_scores(model)
```

Hyperparameter tuning with RandomizedSearchCV

In [182]:

```
%%time
from sklearn.model_selection import RandomizedSearchCV

# Different RandomForestRegressor hyperparameters

rf_grid = {'n_estimators': np.arange(10, 100, 10),
          'max_depth': [None, 3, 5, 10],
          'min_samples_split': np.arange(2, 20, 2),
          'min_samples_leaf': np.arange(1, 20, 2),
           'max_features': [0.5, 1, 'sqrt', 'auto'],
          'max_samples': [10000]}

# instantiate RandomizedSearchCV model
rs_model = RandomizedSearchCV(RandomForestRegressor(n_jobs = -1,
                                                    random_state=42),
                             param_distributions=rf_grid,
                             n_iter = 2,
                             cv=5,
                             verbose=True)
# Fit the RandomizedSearchCV
rs_model.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
Wall time: 3min 24s
Parser   : 186 ms
```

Out[182]:

```
RandomizedSearchCV(cv=5,
                   estimator=RandomForestRegressor(n_jobs=-1, random_state=42),
                   n_iter=2,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'max_features': [0.5, 1, 'sqrt',
                                                         'auto'],
                                        'max_samples': [10000],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 1
9]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 1
8]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 60, 70, 80, 90])},
                   verbose=True)
```

In [183]:
```
# find the best model hyperparameters
rs_model.best_params_
```

Out[183]:
```
{'n_estimators': 80,
 'min_samples_split': 10,
 'min_samples_leaf': 3,
 'max_samples': 10000,
 'max_features': 'sqrt',
 'max_depth': 5}
```

In [184]:
```
# evaluate the RandomizedSearchCV
show_scores(rs_model)
```

Out[184]:
```
{'Training MAE': 11730.789081883375,
 'Valid MAE': 13587.261844750172,
 'Training RMSLE': 0.5045615136209148,
 'Valid RMSLE': 0.5166094651411732,
 'Training R^2': 0.488496697482226,
 'Valid R^2': 0.4867697712132065}
```

Train a model with the best hyperparameters

Note These were found after 100 iterations of `RandomizedSaerchCV`

In [185]:
```
%%time

# most ideals hyperparameters
ideal_model = RandomForestRegressor(n_estimators=40,
                                    min_samples_leaf=1,
                                    min_samples_split=14,
                                    max_features=0.5,
                                    n_jobs=-1,
                                    max_samples=None,
                                    random_state=42)

# fit the ideal model
ideal_model.fit(x_train, y_train)
```

```
Wall time: 1min 58s
```

Out[185]:
```
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=40,
                      n_jobs=-1, random_state=42)
```

In [186]:
```
# scores for ideal_model (trained on all the data)
show_scores(ideal_model)
```

Out[186]:
```
{'Training MAE': 2953.8161137163484,
 'Valid MAE': 5951.247761444453,
 'Training RMSLE': 0.14469006962371858,
 'Valid RMSLE': 0.2452416398953833,
 'Training R^2': 0.9588145522577225,
 'Valid R^2': 0.8818019502450094}
```

In [187]:
```
# scores on rs_model (only trained on ~10,000 examples)
show_scores(rs_model)
```

Out[187]:
```
{'Training MAE': 11730.789081883375,
 'Valid MAE': 13587.261844750172,
 'Training RMSLE': 0.5045615136209148,
 'Valid RMSLE': 0.5166094651411733,
 'Training R^2': 0.488496697482226,
 'Valid R^2': 0.4867697712132065}
```

Make predictions on test data

In [221]:

```
# import the test data
df_test = pd.read_csv('Test.csv',
                      low_memory=False,
                      parse_dates=['saledate'])
df_test.head()
```

Out[221]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate	fiModelDesc	...
0	1227829	1006309	3168	121	3	1999	3688.0	Low	2012-05-03	580G	...
1	1227844	1022817	7271	121	3	1000	28555.0	High	2012-05-10	936	...
2	1227847	1031560	22805	121	3	2004	6038.0	Medium	2012-05-10	EC210BLC	...
3	1227848	56204	1269	121	3	2006	8940.0	High	2012-05-10	330CL	...
4	1227863	1053887	22312	121	3	2005	2286.0	Low	2012-05-10	650K	...

5 rows × 52 columns

In [222]:

```
# Make predictions on the test dataset
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The feature names should m
atch those that were passed during fit. Starting version 1.2, an error will be raised.
Feature names unseen at fit time:
- saledate
Feature names seen at fit time, yet now missing:
- Backhoe_Mounting_is_missing
- Blade_Extension_is_missing
- Blade_Type_is_missing
- Blade_Width_is_missing
- Coupler_System_is_missing
- ...

  warnings.warn(message, FutureWarning)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6476\295568641.py in <module>
      1 # Make predictions on the test dataset
----> 2 test_preds = ideal_model.predict(df_test)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972 
    973         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    581             raise ValueError("No support for np.int64 index based sparse matrices")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately,
**check_params)
    564             raise ValueError("Validation should be done on X, y or both.")
    565         elif not no_val_X and no_val_y:
--> 566             X = check_array(X, **check_params)
    567             out = X
    568         elif no_val_X and not no_val_y:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_la
rge_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_fe
atures, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062 
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065 
   2066     def __array_wrap__(
```

```
ValueError: could not convert string to float: 'Low'
```

Preprocessing the data (getting the test dataset in the same format as our training dataset)

In [223]:
```
df_test.isna().sum()
```

Out[223]:
```
SalesID                         0
MachineID                       0
ModelID                         0
datasource                      0
auctioneerID                    0
YearMade                        0
MachineHoursCurrentMeter    10328
UsageBand                   10623
saledate                        0
fiModelDesc                     0
fiBaseModel                     0
fiSecondaryDesc              3975
fiModelSeries               10451
fiModelDescriptor            9433
ProductSize                  6409
fiProductClassDesc              0
state                           0
ProductGroup                    0
ProductGroupDesc                0
Drive_System                 9698
Enclosure                       2
Forks                        6149
Pad_Type                    10349
Ride_Control                 8216
Stick                       10349
Transmission                 7639
Turbocharged                10349
Blade_Extension             11806
Blade_Width                 11806
Enclosure_Type              11806
Engine_Horsepower           11806
Hydraulics                   2142
Pushblock                   11806
Ripper                       9753
Scarifier                   11806
Tip_Control                 11806
Tire_Size                    9679
Coupler                      4856
Coupler_System              10391
Grouser_Tracks              10391
Hydraulics_Flow             10391
Track_Type                   9063
Undercarriage_Pad_Width      9059
```

```
Stick_Length                9063
Thumb                       9062
Pattern_Changer             9063
Grouser_Type                9063
Backhoe_Mounting           10406
Blade_Type                 10399
Travel_Controls            10399
Differential_Type          10328
Steering_Controls          10328
dtype: int64
```

In [224]:
```
df_test.columns
```

Out[224]:
```
Index(['SalesID', 'MachineID', 'ModelID', 'datasource', 'auctioneerID',
       'YearMade', 'MachineHoursCurrentMeter', 'UsageBand', 'saledate',
       'fiModelDesc', 'fiBaseModel', 'fiSecondaryDesc', 'fiModelSeries',
       'fiModelDescriptor', 'ProductSize', 'fiProductClassDesc', 'state',
       'ProductGroup', 'ProductGroupDesc', 'Drive_System', 'Enclosure',
       'Forks', 'Pad_Type', 'Ride_Control', 'Stick', 'Transmission',
       'Turbocharged', 'Blade_Extension', 'Blade_Width', 'Enclosure_Type',
       'Engine_Horsepower', 'Hydraulics', 'Pushblock', 'Ripper', 'Scarifier',
       'Tip_Control', 'Tire_Size', 'Coupler', 'Coupler_System',
       'Grouser_Tracks', 'Hydraulics_Flow', 'Track_Type',
       'Undercarriage_Pad_Width', 'Stick_Length', 'Thumb', 'Pattern_Changer',
       'Grouser_Type', 'Backhoe_Mounting', 'Blade_Type', 'Travel_Controls',
       'Differential_Type', 'Steering_Controls'],
      dtype='object')
```

In [225]:
```
x_train.columns
```

Out[225]:
```
Index(['SalesID', 'MachineID', 'ModelID', 'datasource', 'auctioneerID',
       'YearMade', 'MachineHoursCurrentMeter', 'UsageBand', 'fiModelDesc',
       'fiBaseModel',
       ...
       'Undercarriage_Pad_Width_is_missing', 'Stick_Length_is_missing',
       'Thumb_is_missing', 'Pattern_Changer_is_missing',
       'Grouser_Type_is_missing', 'Backhoe_Mounting_is_missing',
       'Blade_Type_is_missing', 'Travel_Controls_is_missing',
       'Differential_Type_is_missing', 'Steering_Controls_is_missing'],
      dtype='object', length=102)
```

In [226]:

```
def preprocess_data(df):
    '''
    performs transformations 0on df and returns transformed df.
    '''
    df['saleYear'] = df.saledate.dt.year
    df['saleMonth'] = df.saledate.dt.month
    df['saleDay'] = df.saledate.dt.day
    df['saleDayOfWeek'] = df.saledate.dt.dayofweek
    df['saleDayOfYear'] = df.saledate.dt.dayofyear

    df.drop('saledate', axis = 1, inplace=True)

   # Fill the numeric rows with median
    for label, content in df.items():
        if pd.api.types.is_numeric_dtype(content):
            if pd.isnull(content).sum():
            # Add a binary column which tells us if the data was missing or not
                df[label+'_is_missing'] = pd.isnull(content)
            # Fill missing numeric values with median
                df[label] = content.fillna(content.median())
    # This will turn all of the string value into category values
    for label, content in df.items():
        if pd.api.types.is_string_dtype(content):
            df[label] = content.astype('category').cat.as_ordered()

    # Filled categorical missing data and turn categories into numbers
            if not pd.api.types.is_numeric_dtype(content):
                df[label+'_is_missing'] = pd.isnull(content)
                # we add +1 to the category code because pandas encodes missing catergories as -1
                df[label] = pd.Categorical(content).codes + 1


    return df
```

In [227]:

```
# process test data
df_test = preprocess_data(df_test)
df_test.head()
```

Out[227]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBaseModel
0	1227829	1006309	3168	121	3	1999	3688.0	2	499	180
1	1227844	1022817	7271	121	3	1000	28555.0	1	831	292
2	1227847	1031560	22805	121	3	2004	6038.0	3	1177	404
3	1227848	56204	1269	121	3	2006	8940.0	1	287	113
4	1227863	1053887	22312	121	3	2005	2286.0	2	566	196

5 rows × 101 columns

In [228]:

```
# make predictions on updated test data
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The feature names should m
atch those that were passed during fit. Starting version 1.2, an error will be raised.
Feature names seen at fit time, yet now missing:
- auctioneerID_is_missing

  warnings.warn(message, FutureWarning)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6476\2973351858.py in <module>
      1 # make predictions on updated test data
----> 2 test_preds = ideal_model.predict(df_test)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972 
    973         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    581             raise ValueError("No support for np.int64 index based sparse matrices")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    583 
    584         if not no_val_X and check_params.get("ensure_2d", True):
--> 585             self._check_n_features(X, reset=reset)
    586 
    587         return out

~\anaconda3\lib\site-packages\sklearn\base.py in _check_n_features(self, X, reset)
    398 
    399         if n_features != self.n_features_in_:
--> 400             raise ValueError(
    401                 f"X has {n_features} features, but {self.__class__.__name__} "
    402                 f"is expecting {self.n_features_in_} features as input."

ValueError: X has 101 features, but RandomForestRegressor is expecting 102 features as input.
```

In [229]:

```
x_train.head()
```

Out[229]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBaseModel
0	1646770	1126363	8434	132	18.0	1974	0.0	0	4593	1744
1	1821514	1194089	10150	132	99.0	1980	0.0	0	1820	559
2	1505138	1473654	4139	132	99.0	1978	0.0	0	2348	713
3	1671174	1327630	8591	132	99.0	1980	0.0	0	1819	558
4	1329056	1336053	4089	132	99.0	1984	0.0	0	2119	683

5 rows × 102 columns

In [230]:

```
# we can find how the columns differ using sets
set(x_train.columns) - set(df_test.columns)
```

Out[230]: {'auctioneerID_is_missing'}

In [231]:

```
# Manually adjust df-tesst to have auctioneerID_is_missing
df_test['auctioneerID_is_missing'] = False
df_test.head()
```

Out[231]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBaseModel
0	1227829	1006309	3168	121	3	1999	3688.0	2	499	180
1	1227844	1022817	7271	121	3	1000	28555.0	1	831	292
2	1227847	1031560	22805	121	3	2004	6038.0	3	1177	404
3	1227848	56204	1269	121	3	2006	8940.0	1	287	113
4	1227863	1053887	22312	121	3	2005	2286.0	2	566	196

5 rows × 102 columns

Finally now our test dataframe has the same features as our training dataframe, we can make predictions

In [232]:
```
# make predictions of the test data
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The feature names should m
atch those that were passed during fit. Starting version 1.2, an error will be raised.
Feature names must be in the same order as they were in fit.

  warnings.warn(message, FutureWarning)
```

In [233]:
```
test_preds
```

Out[233]:
```
array([20614.36780887, 19897.80170658, 44852.21959446, ...,
       14296.98620472, 22164.85757662, 31683.80063427])
```

we've made some predictions but they're not in the same format Kaggle is asking for.

In [237]:

```
# Format predictions into the same format kaggle is after
df_preds = pd.DataFrame()
df_preds['SalesID'] = df_test['SalesID']
df_preds['salesPrice'] = test_preds
df_preds
```

Out[237]:

	SalesID	salesPrice
0	1227829	20614.367809
1	1227844	19897.801707
2	1227847	44852.219594
3	1227848	68346.325323
4	1227863	39487.349708
...	...	...
12452	6643171	46466.092910
12453	6643173	17500.493352
12454	6643184	14296.986205
12455	6643186	22164.857577
12456	6643196	31683.800634

12457 rows × 2 columns

In [254]:

```
# Export prediction data
df_preds.to_csv('test_predictions.csv', index=False)
```

Feature Importance

Feature importance seeks to figure out which different attribute of the data were most importance when it comes to predicting the **target varaiable** (SalePrice)

In [255]:

```
# find feature Importance of our best model
ideal_model.feature_importances_
```

Out[255]:

```
array([3.39445533e-02, 1.81148281e-02, 4.09167072e-02, 1.70752171e-03,
       3.40797459e-03, 2.08200698e-01, 2.95067052e-03, 1.10113725e-03,
       4.16122668e-02, 4.71911805e-02, 6.23815431e-02, 4.67433955e-03,
       1.52524442e-02, 1.52517337e-01, 4.72224713e-02, 5.96817956e-03,
       1.29351899e-03, 2.78088439e-03, 2.37248769e-03, 6.17114453e-02,
       8.13525488e-04, 3.61873268e-05, 9.19098115e-04, 2.23170993e-04,
       1.28102678e-03, 2.06519636e-05, 2.01477316e-03, 6.63364759e-03,
       2.15274492e-03, 2.50178165e-03, 4.63902393e-03, 3.85873985e-03,
       2.76062667e-03, 1.00782454e-03, 2.47969268e-04, 6.04239818e-03,
       7.64997072e-04, 1.57100537e-02, 2.29716203e-03, 2.58372272e-03,
       8.07637426e-04, 9.18548690e-04, 1.35656446e-03, 5.81458569e-04,
       4.96716928e-04, 3.79552257e-04, 5.31712788e-04, 2.71823509e-03,
       8.34294376e-04, 3.12136841e-04, 2.14075157e-04, 7.42422919e-02,
       3.80158492e-03, 5.67641024e-03, 2.87154703e-03, 9.83349904e-03,
       2.65470837e-04, 1.57946459e-03, 3.10058108e-04, 0.00000000e+00,
       0.00000000e+00, 2.27421721e-03, 1.05632062e-03, 5.42819222e-03,
       3.48484864e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.90858845e-05, 9.09490682e-06, 1.31265147e-04,
       5.29163902e-06, 1.11952381e-04, 4.78452431e-06, 3.43582863e-04,
       5.57068428e-06, 1.07167376e-03, 3.99179008e-03, 4.07753410e-03,
       1.05749617e-04, 2.76528927e-03, 2.59244312e-05, 3.51888176e-04,
       2.31519337e-03, 1.99211177e-03, 4.02034629e-03, 2.03778082e-04,
       1.13483313e-02, 9.02551628e-04, 1.58182497e-03, 4.63243398e-05,
       2.92071004e-04, 3.11923094e-05, 1.56873538e-04, 2.87205987e-05,
       3.80543083e-05, 2.55045807e-04, 1.66878572e-04, 2.10341792e-04,
       1.26024842e-04, 9.40663015e-05])
```

In [256]:

```
len (ideal_model.feature_importances_)
```

Out[256]: 102

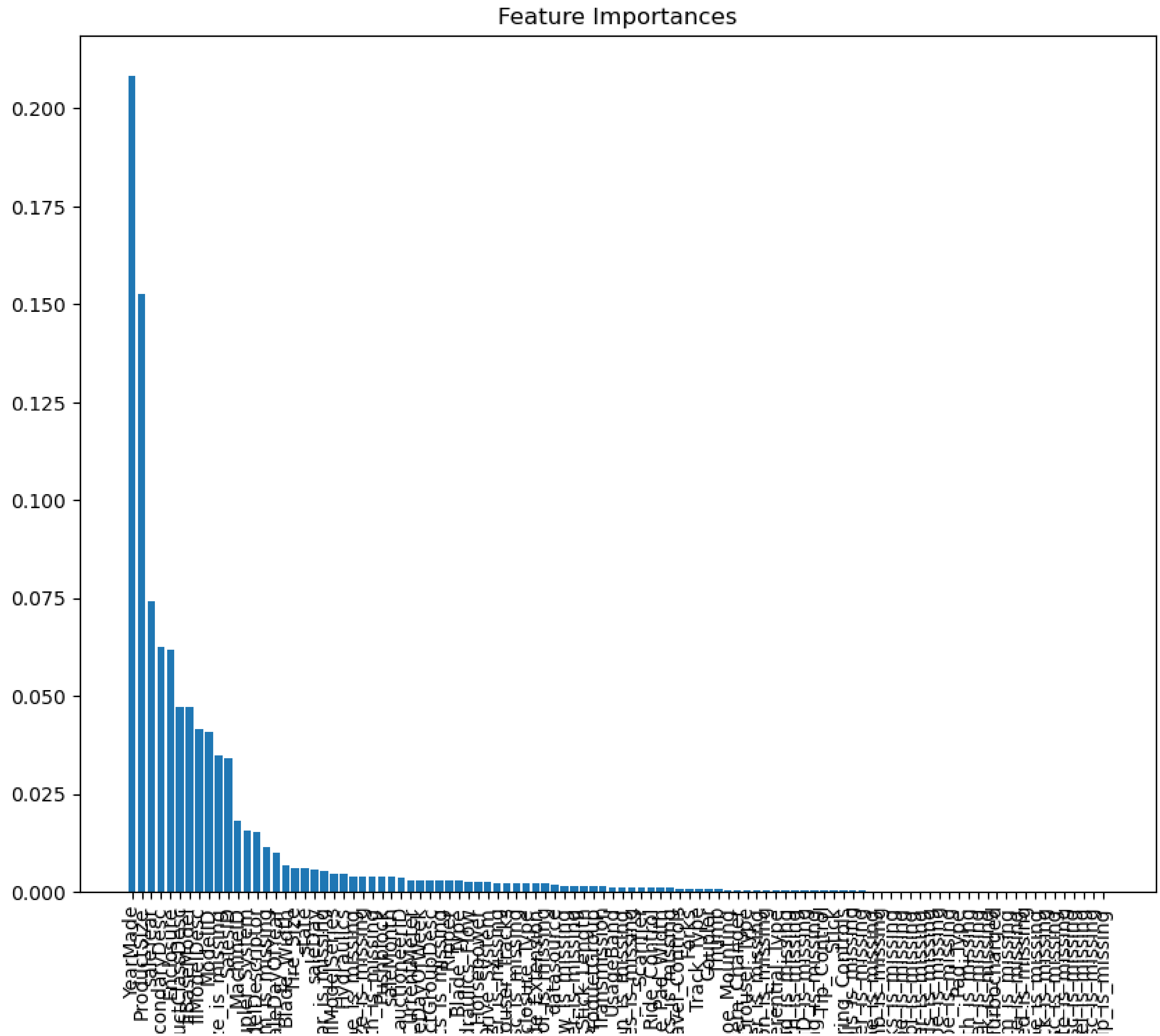
In [296]:

```
#  1.  helper function for plotting feature importances
def plot_feature_importance(ideal_model, x_train):
    importances = ideal_model.feature_importances_
    indices = np.argsort(importances)[: : -1]

    plt.figure(figsize=(10, 8))
    plt.bar(range(x_train.shape[1]), importances[indices])
    plt.xticks(range(x_train.shape[1]), x_train.columns[indices], rotation = 90)
    plt.title('Feature Importances')
    plt.show()

```

In [297]:
```
plot_feature_importance(ideal_model, x_train)
```


Feature Importances
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
YearMade

In [316]:

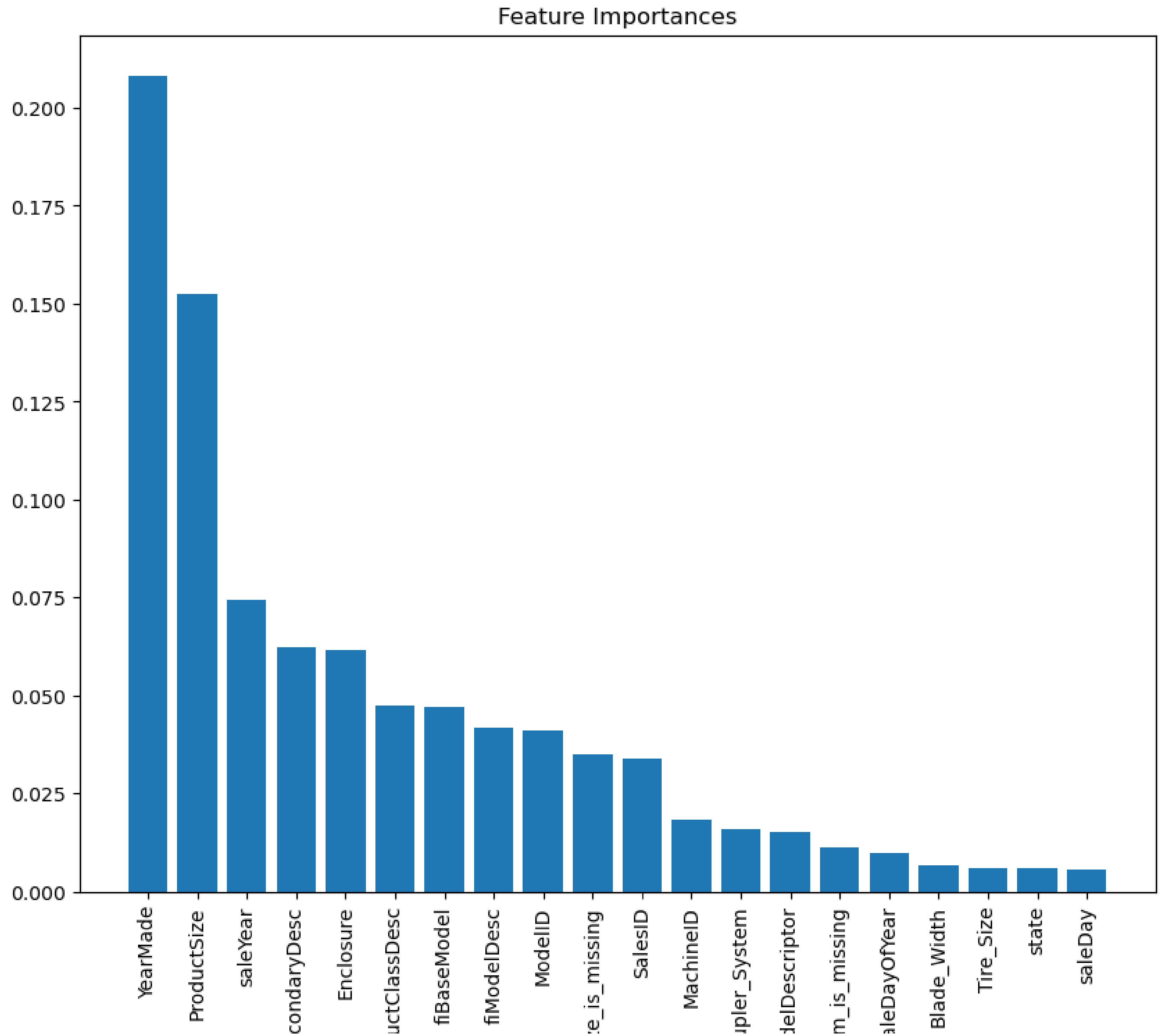
```python
# 2. Helper function for plotting feature importances (Top 20 feature importance)
def plot_feature_importances(ideal_model, x_train):
    importances = ideal_model.feature_importances_
    indices = np.argsort(importances)[: : -1][:20]

    plt.figure(figsize=(10, 8))
    plt.bar(range(len(indices)), importances[indices])
    plt.xticks(range(len(indices)), x_train.columns[indices], rotation = 90)
    plt.title('Feature Importances')
    plt.xticks(rotation=90)
    plt.show()

```

In [317]:

```
plot_feature_importances(ideal_model, x_train)
```

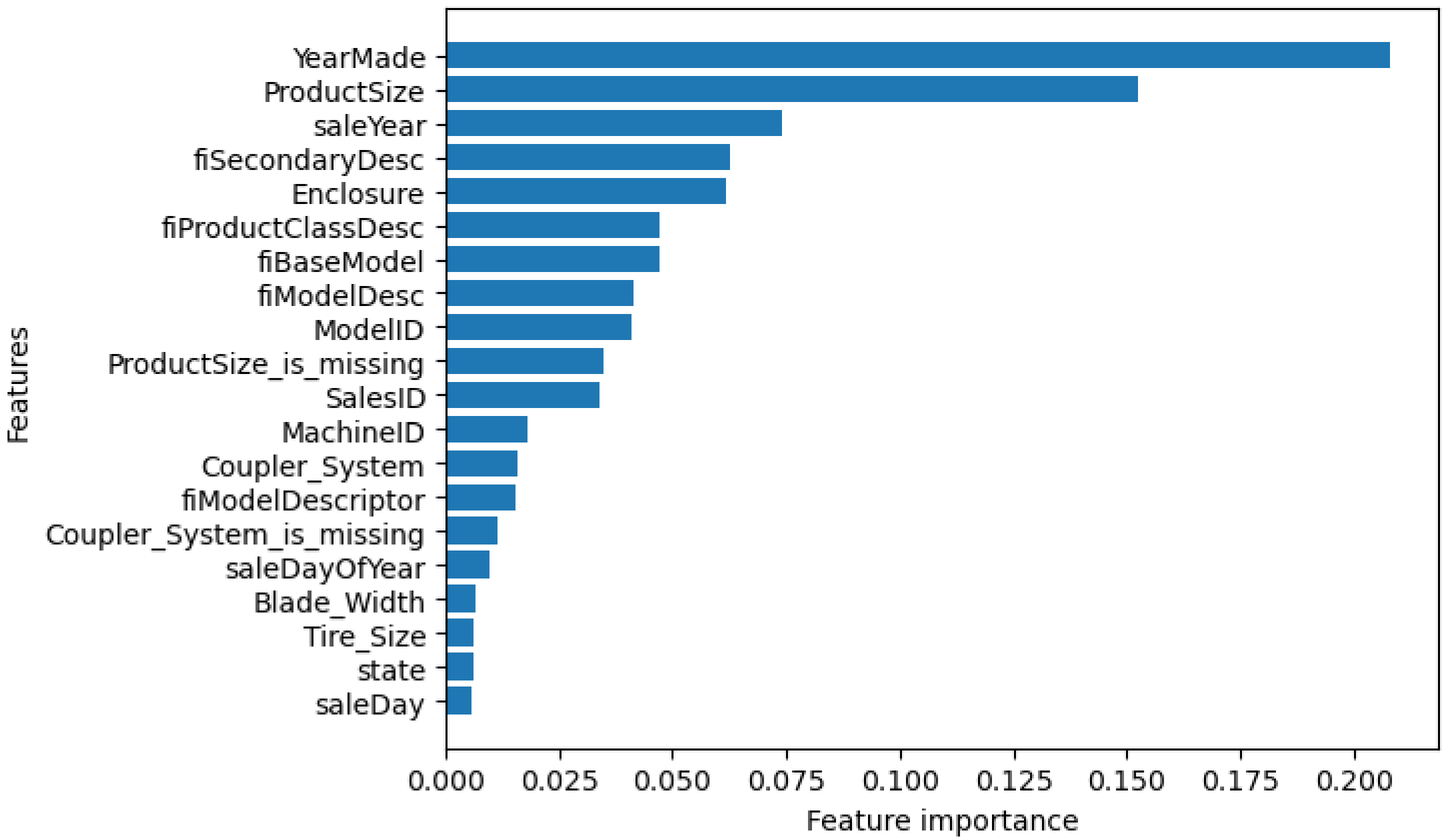

Feature Importances
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
YearMade
ProductSize
saleYear
condaryDesc
Enclosure
uctClassDesc
fiBaseModel
fiModelDesc
ModelID
e_is_missing
SalesID
MachineID
upler_System
delDescriptor
m_is_missing
aleDayOfYear
Blade_Width
Tire_Size
state
saleDay

In [325]:

```
#  3.  helper function for plotting feature importances
def plot_features(columns, importances, n=20):
    df = (pd.DataFrame({'features': columns,
                        'feature_importances': importances})
          .sort_values('feature_importances', ascending=False)
          .reset_index(drop=True))
#     plot the dataframe
    fig, ax = plt.subplots()
    ax.barh(df['features'][:n], df['feature_importances'][:20])
    ax.set_ylabel('Features')
    ax.set_xlabel('Feature importance')
    ax.invert_yaxis()
```

In [326]:

```
plot_features(x_train.columns, ideal_model.feature_importances_)
```



Question to finish: Why might knowing the feature importances of a trained machine learning model be helpful?

Final challenge what other machine learning models counld you try on our dataset?

In [ ]:

```

```